Stochastic Simulation of Process Calculi for Biology

Andrew Phillips
Microsoft Research
Cambridge, United Kingdom
andrew.phillips@microsoft.com

Matthew R. Lakin
Microsoft Research
Cambridge, United Kingdom
v-mlakin@microsoft.com

Loïc Paulevé
IRCCyN, UMR CNRS 6597
École Centrale de Nantes, France
loic.pauleve@irccyn.ec-nantes.fr

Biological systems typically involve large numbers of components with complex, highly parallel interactions and intrinsic stochasticity. To model this complexity, numerous programming languages based on process calculi have been developed, many of which are expressive enough to generate unbounded numbers of molecular species and reactions. As a result of this expressiveness, such calculi cannot rely on standard reaction-based simulation methods, which require fixed numbers of species and reactions. Rather than implementing custom stochastic simulation algorithms for each process calculus, we propose to use a generic abstract machine that can be instantiated to a range of process calculi and a range of reaction-based simulation algorithms. The abstract machine functions as a just-in-time compiler, which dynamically updates the set of possible reactions and chooses the next reaction in an iterative cycle. In this short paper we give a brief summary of the generic abstract machine, and show how it can be instantiated with the stochastic simulation algorithm known as Gillespie's Direct Method. We also discuss the wider implications of such an abstract machine, and outline how it can be used to simulate multiple calculi simultaneously within a common framework.

1 Introduction

Biological systems typically involve large numbers of components with complex, highly parallel interactions and intrinsic stochasticity. To model this complexity, numerous programming languages based on process calculi have been developed, many of which are expressive enough to generate unbounded numbers of molecular species and reactions. Examples include variants of the stochastic pi-calculus [13, 14, 11], BlenX [3], the kappa calculus [2], and variants of the bioambient calculus [15, 10]. As a result of this expressiveness, such calculi cannot rely on standard reaction-based simulation methods such as [7, 5], which require fixed numbers of species and reactions. Instead, a custom simulation algorithm is typically developed for each calculus. The choice of algorithm depends on the nature of the underlying biological system, such as whether exact simulation is required [6, 5], whether certain reactions operate at different timescales [7, 16], or whether non-Markovian reaction rates are needed [1, 8].

Rather than implementing custom stochastic simulation algorithms for each process calculus, we propose to use a generic abstract machine that can be instantiated to a range of process calculi and a range of reaction-based simulation algorithms. The abstract machine functions as a just-in-time compiler, which dynamically updates the set of possible reactions and chooses the next reaction in an iterative cycle. The abstract machine is instantiated to a particular calculus by defining two functions: one for transforming a process of the calculus to a set of species, and another for computing the set of possible reactions between species. The abstract machine is instantiated to a particular simulation algorithm by definition three functions: one for computing the next reaction, one for computing the reaction activity from an initial set of reactions and species populations, and a third for updating the reaction activity as the species populations change over time. Having a clear separation between the simulation algorithm and the language specification allows us not only to easily instantiate the machine to different process calculi,

G.Ciobanu, M.Koutny (Eds.): Membrane Computing and
Biologically Inspired Process Calculi 2010 (MeCBIC 2010)
EPTCS 40, 2010, pp. 1–5, doi:10.4204/EPTCS.40.1

Table 1: Syntax of the generic abstract machine, where a term T consists of the current time t, a species map S and a reaction map R. We let $\widetilde{I}$ denote a multiset of species $\{I_1,..,I_N\}$ and $\widetilde{O}$ denote a set of reactions.

T	syntax	description
T	(t,S,R)	Time t, species map S, reaction map R
S	$\{I_1 \mapsto i_1,..,I_N \mapsto i_N\}$	Map from a species I to its population i,
R	$\{O_1 \mapsto A_1,..,O_N \mapsto A_N\}$	Map from a reaction O to its activity A
O	$(\widetilde{I},r,\widetilde{I'})$	Reaction with reactants $\widetilde{I}$, products $\widetilde{I'}$ and rate r.

Table 2: Parameterised definition of the generic abstract machine. If $\widetilde{I}$ is a multiset $\{I_1,..,I_N\}$ we write $\widetilde{I} \oplus T$ for $I_1 \oplus .. \oplus I_N \oplus T$, and $T \ominus \widetilde{I}$ for $T \ominus I_1 \ominus .. \ominus I_N$ (the order is unimportant). We write $dom(S)$ for the domain of S. We also write $S(I)$ for the value associated with I in S, and $S\{I \mapsto v\}$ for S updated so that v is associated with I.

function		definition
$P \oplus T$	$\triangleq$	$species(P) \oplus T$
$I \oplus (t,S,R)$	$\triangleq$	$(t,S',R \cup R')$ if $\widetilde{I'} = dom(S)$; $I \notin \widetilde{I'}$; $S' = S\{I \mapsto 1\}$;
		$\widetilde{O} = reactions(I,\widetilde{I'})$; $R' = init(\widetilde{O},(t,S',R))$
$I \oplus (t,S,R)$	$\triangleq$	$(t,S',R \cup R')$ if $S(I) = i$; $S' = S\{I \mapsto i+1\}$; $R' = updates(I,(t,S',R))$
$(t,S,R) \ominus I$	$\triangleq$	$(t,S',R \cup R')$ if $S(I) = i$; $S' = S\{I \mapsto i-1\}$; $R' = updates(I,(t,S',R))$

but also to add new simulation algorithms that can be shared between calculi. Furthermore, the approach could be used to dynamically integrate the simulation of multiple process calculi simultaneously, acting as a common language runtime for the simulation of process calculi for biology.

In this short paper we give a brief summary of the generic abstract machine of [8], and show how it can be instantiated with the stochastic simulation algorithm of [6]. We also discuss the wider implications of such an abstract machine, and outline how it can be used to simulate multiple calculi simultaneously within a common framework.

2 Summary of the Abstract Machine

The syntax of the generic abstract machine is summarised in Table 1, and is based on the definitions of [8]. A machine term T is a triple (t,S,R), where t is the current time, S is a map from a species I to its population i, and R is a map from a reaction O to its activity A, which is used to compute the next reaction. Each reaction is represented by a tuple $(\widetilde{I},r,\widetilde{I'})$, where $\widetilde{I}$ denotes the multiset of reactant species, $\widetilde{I'}$ denotes the multiset of product species and r denotes the reaction rate. The structure of a term of the abstract machine can be summarised as follows.

Machine term T				
Time t	Species map S		Reaction map R	
	Species	Population	Reaction	Activity
	I_1	i_1	$\widetilde{I_1} \xrightarrow{r_1} \widetilde{I'_1}$	A_1
	...	...	...	...
	I_N	i_N	$\widetilde{I_M} \xrightarrow{r_M} \widetilde{I'_M}$	A_M

Table 3: Instantiation of the generic abstract machine with the stochastic simulation algorithm of [6]. We write $\{E_i \mid C_1;..;C_N\}$ to denote the set of elements E_i that satisfy conditions $C_1;..;C_N$. We let n_1 and n_2 denote two random numbers from the standard uniform distribution, $U(0,1)$. The function $propensity(O,S)$ is defined in the main text.

function		**definition**
$next(t,S,R)$	$\triangleq$	$O_\mu, t+t'$ if $a_0 = \sum_{O_i \in dom(R)} R(O_i)$; $t' = (\frac{1}{a_0}) ln(\frac{1}{n_1})$; $\sum_{i=1}^{\mu-1} a_i < n_2 a_0 \leq \sum_{i=1}^{\mu} a_i$
$init(\widetilde{O},(t,S,R))$	$\triangleq$	$\{O_i \mapsto propensity(O_i,S) \mid O_i \in \widetilde{O}\}$
$updates(I,(t,S,R))$	$\triangleq$	$\{O_i \mapsto propensity(O_i,S) \mid O_i \in dom(R); O_i = (\tilde{J},r,\tilde{J'}); I \in \tilde{J}\}$

To instantiate the abstract machine with a given process calculus, it is sufficient to define a function $species(P)$ for transforming a process P to a multiset of species, together with a function $reactions(I,\widetilde{I'})$ for computing the set of reactions between a new species I and an existing set of species $\widetilde{I'}$. The syntax of species I is specific to the choice of process calculus. The *species* function is used to initialise the abstract machine at the beginning of a simulation, while the *reactions* function is used to update the set of possible reactions dynamically. This allows systems with potentially unbounded numbers of species and reactions to be simulated.

To instantiate the abstract machine with a given simulation algorithm, it is sufficient to define a function $next(T)$ for choosing the next reaction from a term T, a function $init(\widetilde{O},T)$ for initialising a term with a set of reactions $\widetilde{O}$, and a function $updates(I,T)$ for updating the reactions in a term affected by a given species I. The abstract machine is then executed by repeated application of the following rule.

$$\frac{(\widetilde{I},r,\widetilde{I'}),t' = next(t,S,R)}{t,S,R \xrightarrow{(\widetilde{I},r,\widetilde{I'})} \widetilde{I'} \oplus ((t',S,R) \ominus \widetilde{I})}$$

Each time the next reaction is selected, it is executed by removing the reactants $\widetilde{I}$ from the machine term, adding the products $\widetilde{I'}$ and updating the current time of the machine. Corresponding definitions for adding and removing species are summarised in Table 2. A process P is added to a machine term T by computing the multiset of species $\{I_1,\ldots,I_N\}$ which correspond to P and then adding each of these species to the term. If a new species I is already present in the term then its population is incremented in S and the activity of the affected reactions is updated. If the species is not already present in the term, its population is set to 1 in S and new reactions for the species are computed, together with their activity. The operation $T \ominus \widetilde{I}$ removes the species $\widetilde{I}$ from the machine term T, by decrementing the corresponding species populations and by updating the affected reactions.

3 Instantiating the Abstract Machine

An instantiation of the abstract machine with the stochastic simulation algorithm of [6] is outlined in Table 3. Each reaction $(\widetilde{I},r,\widetilde{I'})$ is mapped to its propensity a_i, which is computed by multiplying the rate of the reaction by the number of distinct combinations of the reactants $\widetilde{I}$. The function $propensity(O,S)$ computes the propensity of the reaction O given the species map S and is defined as follows, assuming

that reactions are either unary or binary and that I_1 and I_2 are distinct species.

$$\begin{aligned} \mathit{propensity}((\{I_1\}, r, \widetilde{I'}), S) &\triangleq r \times S(I_1) \\ \mathit{propensity}((\{I_1, I_1\}, r, \widetilde{I'}), S) &\triangleq r \times S(I_1) \times (S(I_1) - 1)/2 \\ \mathit{propensity}((\{I_1, I_2\}, r, \widetilde{I'}), S) &\triangleq r \times S(I_1) \times S(I_2) \end{aligned}$$

The function $\mathit{init}(\widetilde{O}, T)$ computes the initial propensity for each reaction in $\widetilde{O}$, using the initial species populations in T, while the function $\mathit{updates}(I, T)$ updates the propensities of all the reactions in T for which I is a reactant. Finally, the function $\mathit{next}(T)$ chooses a reaction from T with probability proportional to the reaction propensity, and computes the corresponding duration of the reaction according to [6].

We have also instantiated the abstract machine to the Next Reaction Method of [5] and to the Non-Markovian Next Reaction Method of [8], by defining corresponding *init*, *next* and *updates* functions, as described in [8]. We have used the abstract machine to implement the DNA Strand Displacement (DSD) calculus for modelling DNA circuits [12], the Genetic Engineering of Cells (GEC) calculus for modelling of genetic devices [9], and the Stochastic Pi Machine (SPiM) calculus for general modelling of biological systems [17], by defining appropriate *species* and *reactions* functions for each calculus. Simulators for these three calculi are available online at `http://research.microsoft.com/dna`, `http://research.microsoft.com/gec` and `http://research.microsoft.com/spim`, respectively. Technical details of the instantiation of the generic abstract machine with the stochastic pi-calculus and the bioambient calculus are outlined in [8]. We are currently developing an instantiation of the generic abstract machine to the kappa calculus of [2]. Although the idea of integrating different modelling and simulation methods within a common framework is not a new one [4], our approach is the first attempt to formally define a generic framework for simulating a broad range of process calculi with an arbitrary reaction-based simulation algorithm.

4 A Common Simulation Framework

The generic abstract machine can be used to simulate multiple calculi simultaneously by assuming a separate species type I_L for each calculus L, together with an initial set of cross-calculus reactions $\widetilde{O_0}$. An example of a cross-calculus reaction is $I_{DSD} + I_{SPiM} \xrightarrow{r} I_{SPiM} + I'_{SPiM}$, which takes a species of the DSD language, such as a known DNA vaccine assembled via strand displacement, together with a species of the SPiM language, such as a polymerase, and produces a corresponding protein species in SPiM together with the original polymerase. The reaction therefore enables the output of a strand displacement model in DSD to interface with a cellular model in SPiM. For each dynamically created species I_L the function $\mathit{reactions}(I_L, \widetilde{I'})$ calls the appropriate calculus-specific function $\mathit{reactions}_L(I_L, \widetilde{I'_L})$, where $\widetilde{I'_L}$ denotes the subset of species in $\widetilde{I'}$ that are of type L. This approach allows multiple calculi to interact with each other within the same simulation environment, via a fixed set of interface reactions. Further work is needed to formalise the multi-language execution paradigm in more detail.

The generic abstract machine can therefore be used to simulate a range of existing process calculi within a common framework. By decoupling the choice of calculus from the choice of simulation algorithm, multiple calculi can re-use the same algorithm via a common interface, without the need to implement custom simulation algorithms for each calculus. In future, this could allow models to be constructed from components written in different domain-specific languages, each designed to allow a natural, concise encoding of that component. The components could then interact dynamically via a common language runtime, allowing integrated simulation of heterogeneous biological systems.

References

[1] Dmitri Bratsun, Dmitri Volfson, Lev S. Tsimring & Jeff Hasty (2005): *Delay-induced stochastic oscillations in gene regulation*. Proceedings of the National Academy of Sciences of the United States of America 102(41), pp. 14593–14598. Available at http://www.pnas.org/content/102/41/14593.abstract.

[2] Vincent Danos, Jéréme Feret, Walter Fontana, Russell Harmer & Jean Krivine (2007): *CONCUR 2007 - Concurrency Theory*, chapter Rule-Based Modelling of Cellular Signalling, pp. 17–41. Available at http://dx.doi.org/10.1007/978-3-540-74407-8_3.

[3] Lorenzo Dematté, Corrado Priami & Alessandro Romanel (2008): *Modelling and simulation of biological processes in BlenX*. SIGMETRICS Performance Evaluation Review 35(4), pp. 32–39. Available at http://doi.acm.org/10.1145/1364644.1364653.

[4] Roland Ewald, Jan Himmelspach, Matthias Jeschke, Stefan Leye & Adelinde M Uhrmacher (2010): *Flexible experimentation in the modeling and simulation framework JAMES II – implications for computational systems biology*. Brief Bioinform Available at http://dx.doi.org/10.1093/bib/bbp067.

[5] Michael A. Gibson & Jehoshua Bruck (2000): *Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels*. The Journal of Physical Chemistry A 104(9), pp. 1876–1889. Available at http://pubs.acs.org/doi/abs/10.1021/jp993732q.

[6] Daniel T. Gillespie (1977): *Exact stochastic simulation of coupled chemical reactions*. J. Phys. Chem. 81(25), pp. 2340–2361.

[7] Daniel T. Gillespie (2001): *Approximate accelerated stochastic simulation of chemically reacting systems*. J. Chem. Phys. 115, pp. 1716–1733.

[8] Loïc Paulevé, Simon Youssef, Matthew R. Lakin & Andrew Phillips (2010): *A generic abstract machine for stochastic process calculi*. In: CMSB '10: Proceedings of the 8th International Conference on Computational Methods in Systems Biology, ACM, New York, NY, USA, pp. 43–54.

[9] Michael Pedersen & Andrew Phillips (2009): *Towards programming languages for genetic engineering of living cells*. Journal of the Royal Society Interface 6(S4), pp. 437–450.

[10] Andrew Phillips (2009): *An Abstract Machine for the Stochastic Bioambient Calculus*. Electronic Notes in Theoretical Computer Science 227, pp. 143–159.

[11] Andrew Phillips & Luca Cardelli (2007): *Efficient, Correct Simulation of Biological Processes in the Stochastic Pi-calculus*. In: Computational Methods in Systems Biology, LNCS 4695, Springer, pp. 184–199.

[12] Andrew Phillips & Luca Cardelli (2009): *A programming language for composable DNA circuits*. Journal of the Royal Society Interface 6(S4), pp. 419–436.

[13] C. Priami, A. Regev, E. Shapiro & W. Silverman (2001): *Application of a stochastic name-passing calculus to representation and simulation of molecular processes*. Information Processing Letters 80, pp. 25–31.

[14] A. Regev, W. Silverman & E. Shapiro (2001): *Representation and Simulation of Biochemical Processes Using the pi-Calculus Process Algebra*. In: Pacific Symposium on Biocomputing, 6, World Scientific Press, Singapore, pp. 459–470.

[15] Aviv Regev, Ekaterina M. Panina, William Silverman, Luca Cardelli & Ehud Y. Shapiro (2004): *BioAmbients: an abstraction for biological compartments*. Theor. Comput. Sci. 325(1), pp. 141–167. Available at http://dx.doi.org/10.1016/j.tcs.2004.03.061.

[16] Tianhai Tian & Kevin Burrage (2004): *Binomial leap methods for simulating stochastic chemical kinetics*. J. Chem. Phys. 121, pp. 10356–10364.

[17] Dennis Y. Q. Wang, Luca Cardelli, Andrew Phillips, Nir Piterman & Jasmin Fisher (2009): *Computational modeling of the EGFR network elucidates control mechanisms regulating signal dynamics*. BMC Systems Biology 3(118).